Laboratorio Nro. 3
Vuelta atrás (Backtracking)

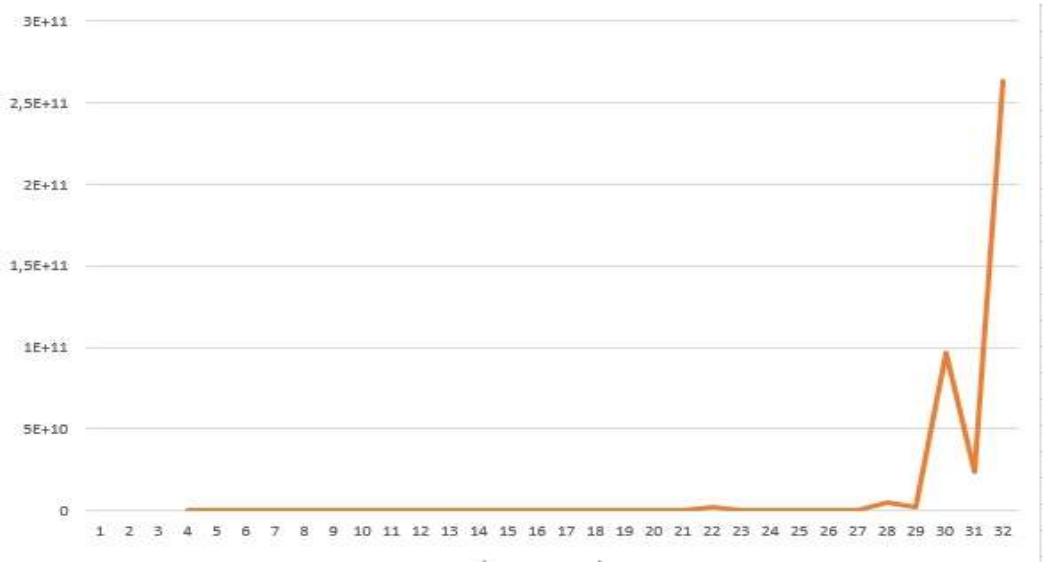
Manuela Valencia Toro
Universidad Eafit
Medellín, Colombia
mvalenciat@eafit.edu.co

Laura Sánchez Córdoba
Universidad Eafit
Medellín, Colombia
lsancheszc@eafit.edu.co

Felipe Olaya Ospina
Universidad Eafit
Medellín, Colombia
folayao@eafit.edu.co

1.8) Este algoritmo usa primordialmente los Arreglos y ArrayList como estructuras de datos para su funcionamiento dado que el acceso a estos el más fácil y es lo que se requiere. El algoritmo inicia con un arreglo de distancias (costos) infinitos que se irá llenando con los pesos de recorridos y un ArrayList auxiliar que irá guardando cada nodo recorrido. Mediante el uso del algoritmo recursivo DFS se irán agregando los nodos sucesores con menor peso ya que recursivamente se van comparando estos a medida que el grafo sea recorrido hasta que se llegue al nodo objetivo, de ahí se retorna un ArrayList con la solución.

3) Simulacro de preguntas de sustentación de Proyectos

1. Además de estos existen otros algoritmos basados en buscar la ruta más corta entre dos nodos basados en sus distancias como en Algoritmo de Dijsktra, Bellman-Ford (el cual permite también distancias negativas), el algoritmo de Johnson que lo hace para cada par de vértices y otros algoritmos voraces y de programación dinámica.
2. 

Número de reinas	Tiempo en nanosegundos	Tiempo en nanosegundos de Nreinas con fuerza bruta
4	290056	3000000
5	303579	1100000
6	344149	1400000
7	379124	6200000
8	883690	237000000
9	1184005	1601000000
10	1165818	51312000000
11	1414371	
12	1752459	
13	2065831	
14	2488789	
15	2749467	
16	7541447	
17	847210	
18	3624903	
19	3935337	
20	180012647	

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

21	11590102	
22	1800292014	
23	32749709	
24	490168885	
25	64553240	
26	536806758	
27	645795405	
28	4537700241	
29	2466933189	
30	97079687896	
31	23752479930	
31	2,62899E+11	

Se evidencia una enorme diferencia, cabe aclarar que es debida a los siguientes factores:

- En el algoritmo de fuerza bruta se piden todas las soluciones y su impresión.
- En este ejercicio se pide solo retornar una solución
- Los datos en el caso de fuerza bruta solo se pudieron tomar hasta el 11 debido a problemas de memoria.
- Los datos de la tercera columna son un aproximado de la conversión debido a que se hallaban en unidades diferentes.

3.

Ambas técnicas constituyen métodos sistemáticos para visitar todos los vértices y arcos del grafo, exactamente una vez y en un orden específico predeterminado, por lo cual podríamos decir que estos algoritmos simplemente nos permiten hacer recorridos controlados dentro del grafo con algún propósito. Sin embargo, si se desea hacer una búsqueda con cada nivel del grafo, es decir partiendo de la raíz y los sucesores de la raíz se usa BFS. Y si se requiere partir de la raíz hasta las hojas pasando por cad uno de los sucesores de la raíz y posteriormente los sucesores de Estos se usa DFS.

Tomado de: http://www.bibliadelprogramador.com/2014/04/algoritmos-de-busqueda-en-anchura-bfs-y.html

4.

Algoritmo de Búsqueda A

Conocido también como A asterisco o A estrella fue presentado por Peter E. Hart, Nils J. Nilsson y Bertram Raphael en el año 1968,se clasifica dentro de los algoritmos de búsqueda en grafos. Su función es encontrar siempre y cuando se cumplan determinadas condiciones, el camino de menor costo entre un nodo origen y uno

objetivo, es la forma más ampliamente conocida de la búsqueda primero el mejor, siendo la búsqueda A* tanto completa como óptima.
También está el algoritmo de Dijkstra que se basa en hallar el camino más corto entre dos vértices basado en el coso o suma total de las distancias y otros ya mencionados en el numeral 3.1
Tomado de: https://en.wikipedia.org/wiki/Shortest_path_problem#Algorithms

5. Para empezar, el algoritmo recibe datos desde la consola mediante un BufferedReader, estos corresponden al tamaño del grafo, su número de arcos y las singularidades de cada uno de estos (origen, destino y peso). A partir de la asignación de estos valores a diferentes variables se crea el grafo con estas características y se envía al método recorrer() el cual está basado en una implementación del algoritmo DFS que parte del primer nodo (1 por defecto).
   En el método recorrer se crea un arreglo booleano de visitados (el cual impedirá que el algoritmo recorra por nodos que anteriormente ya ha visitado); a su vez dos ArrayList una para la respuesta y otra auxiliar que guardará los recorridos recursivamente he irá comparando el peso de cada uno hasta hallar el más pequeño, estos pesos son almacenados en un arreglo distancias el cual es inicializado con valor infinito hasta que se y vayan añadiendo los valores. Cabe resaltar que el uso de estas estructuras es dado por la facilidad de acceso a la hora de necesitar un dato específico.
   El método recorrer tiene a su vez un auxiliar del mismo nombre el cual es el encargado de hallar las rutas, parte de un nodo y examina sus sucesores, los agrega y va sacando de la lista a medida que examina los pesos hasta llegar al objetivo de la manera más óptima.
   Finalmente se retorna una lista con la respuesta, si la hay se imprime de lo contrario se imprime -1

6.

```
public static ArrayList recorrer (DigraphAL g, int inicio, int objetivo){

        boolean[] visitados = new boolean[g.size];   //C

        if (inicio == objetivo){ //C

        return null; //C


        }

        ArrayList <Integer> resp = new ArrayList <>(); //C

        ArrayList <Integer> aux = new ArrayList<>(); //C

        int distancias[] = new int[g.size]; //C

        for (int i = 0; i < distancias.length; i++) { C * n

    distancias[i] = Integer.MAX_VALUE; C*n

        }
```

```
        resp = recorrer(g,visitados,inicio,objetivo,resp,distancias,aux,0); //O(m^n)

    resp.add(0,1);

        return resp; //C

}

 public static ArrayList recorrer (DigraphAL g, boolean[] visitados, int v, int f,ArrayList resp,int[] distancias,ArrayList aux,int

pes){

 if (v == f){
        if (pes < distancias[0]){

         distancias[0] = pes; //C

        resp = (ArrayList)aux.clone(); //C

        return resp; //C

        }

        }

        else if (pes > distancias[0]){

        return resp; //C

        }

        try {

      ArrayList<Integer> sucesores = g.getSuccessors(v); //C

      if(sucesores != null){

        visitados[v-1] = true; //C

         for(Integer sucesor: sucesores) {

            aux.add(sucesor); //C* n
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

```
                resp = recorrer(g, visitados, sucesor, f, resp, distancias, aux, pes + g.getWeight(v, sucesor)); // C *
T(m-1)*n

                Integer a = sucesor; //C*n

                aux.remove(a);  //C*n

            }

             visitados[v-1] = false; //C

            }

            return resp; //C

            }catch (Exception e){

            return resp; //C

            }

   }

}
```

T(n)=C+T(n-1)*m
O(n,m)=O(m^n)

7. Las variables m y n significan el número de nodos recorridos recursivamente para hallar la ruta más corta y el número de arcos tomados en cuenta para estas respectivamente.

4) Simulacro de Parcial

1. A)int res = solucionar(n - a, a, b, c) + 1;
   B)res = max(res, solucionar(n - b, a, b, c) + 1
   C)res = max(res, solucionar(n -c, a, b, c) + 1)
2. A)if (pos==graph.length)
   B)if(sePuede(v,graph,path,pos+1)
   C)if(cicloHamilAux(graph,path,pos+1)

3. A) 0-3-7-4-2-6
   1-2-4-6
   2-4-6
   3-7
   4-2-6
   5

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

6-2-4
7
B) 0-3-4-7-2-6
1-2-5-4-6
2-4-6
3-7
4-2-6
5
6-2-4
7

4.
```
public static ArrayList<Integer> hayCamino (Digraph g,int p, int q){
boolean pasos[]= new boolean[g.size()];
ArrayList <Integer>camino=new ArrayList<Integer>();
 Aux(g,nodo, objetivo,pasos,camino);
return camino
}
static boolean a=false;
private static boolean Aux(Digraph g, int nodo, int objetivo, boolean[] visitados, ArrayList<Integer>
list) {

visitados[nodo] = true;
list.add(nodo);
ArrayList<Integer> sucesores = g.getSuccessors(nodo);
if(nodo==objetivo){
a=true;
return true;
        }
if (sucesores != null){
for(Integer sucesor: sucesores){
if (!visitados[sucesor])// vi[sucesor] != false
  Aux(g,sucesor,objetivo,  visitados,list);

}}
return a;

}
```

5. A) 1
B)Línea 11: ni, nj
C)T(n) = t(n-1) +c

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co